القوائم Lists

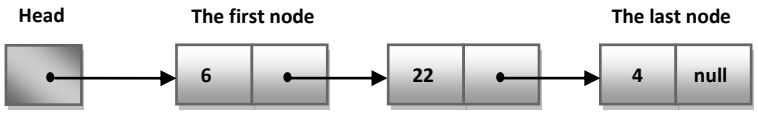
4. القائمة المرتبطة Linked list:

1.4 التاريخ History :

القائمة المرتبطة linked list قدمت سنة 1955– 1956 من قبل ,Allen Newell و Cliff Shaw و Herbert Simon في شركة راند RAND Corporation كتركيبة بيانات أساسية Information Processing Language للغات برمجة المعلومات.

2.4 التعريف Definition :

القائمة المرتبطة linked list هي تركيبة بيانات data structure تتضمن مجموعة من العقد nodes التي تشكل مع بعض سلسلة. في قائمة مرتبطة liked list كل عقدة node تتكون من حقلين : الأول هو البيانات data والثاني هو الوصلة link للعقدة التالية next node . العقدة الأخيرة last node تبين نهاية القائمة.



القوائم المرتبطة linked list تعتبر من بين تراكيب البيانات الأسهل و الأكثر شيوعاً، وهي تستخدم لتطبيق عدة تراكيب بيانات مجردة abstract data type من هذه التراكيب: المكدس stack والطابور queue وغيرها...

بما تتميز القائمة المرتبطة linked list عن المصفوفة array ؟.

- في القائمة المرتبطة linked list عناصر هذه القائمة items يمكن بسهولة إضافتها inserted أو إزالتها removed بدون إعادة تخصيص reallocation أو إعادة تنظيم reorganization للتركيبة بالكامل لأن

عناصر البيانات data items ليس من الضروري أن تخزن بشكل متتالي في الذا كرة memory أو في القرص disk.

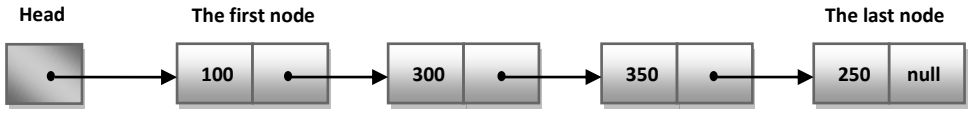
- القائمة المرتبطة linked list تسمح بإضافة insertion وإزالة removal العقد nodes بكفاءة من أي موقع position في القائمة وذلك باستخدام دالة تخصيص الذاكرة the malloc function.

دالة تخصيص الذاكرة The malloc Function :

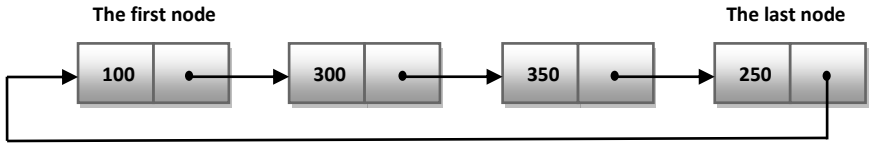
وهي دالة تطلب من النظام حيز من الذاكرة (كتلة block) بحجم محدد، ويتم الحصول على الحيز مع مؤشر pointer يؤشر لأول عنصر في الحيز (الكتلة block).

3.4 القائمة المرتبطة الخطية والدائرية Linear and circular linked list :

القائمة المرتبطة الخطية linear linked list هي القائمة التي يكون فيها العقدة الأخيرة last node تشير إلى لا شئ null أي (next node = null).



القائمة المرتبطة الدائرية circular linked list هي القائمة التي يكون فيها العقدة الأخيرة last node تشير إلى العقدة الأولى first node (next node = address of first node).



4.4 العمليات التي تجرى على القائمة المرتبطة The operations of linked list :

العمليات الأساسية التي تنجز على القائمة المرتبطة هي:

1. إضافة addition.
2. إلغاء deletion.
3. بحث searching.

5.4 أنواع القوائم المرتبطة Types of Linked Lists :

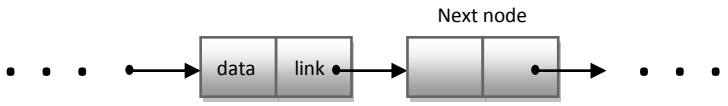
1. القائمة المرتبطة الأحادية single-linked list .
2. القائمة المرتبطة الثنائية Double-linked list .

1.5.4 القائمة المرتبطة الأحادية Single Linked List:

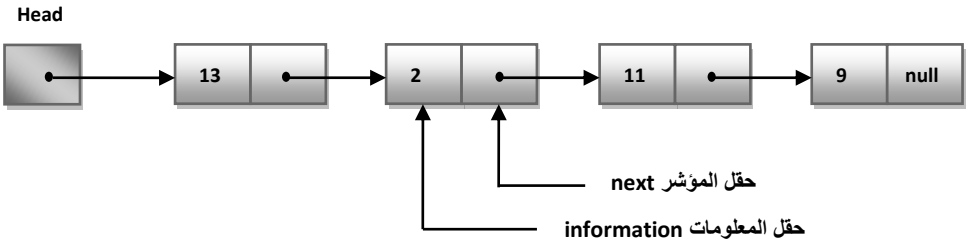
المفاهيم الأساسية والأسماء التعريفية Basic concepts and nomenclature :

كل سجل record أو عقدة node يتضمن حقلين two fields :

- **الحقل الأول the first field :** يتضمن معلومات عن العنصر item ويعرف **بالبيانات data** أو **المعلومات information** أو **القيمة value**.
- **الحقل الثاني the second field :** يتضمن العنوان address للعقدة التالية next node في القائمة وهو عادةً يدعى **الوصلة link** أو **المؤشر pointer** .



المؤشر الرأسي head pointer : يستخدم ليحمل العنوان address لأول عنصر في القائمة list . أيضاً، العنصر الأخير للقائمة المرتبطة يحمل قيمة null في حقل next pointer ليدل على نهاية القائمة list .



إعلان القائمة المرتبطة الأحادية Declaration of a single Linked List:

نفرض أننا نريد تخزين قائمة list من الأعداد الصحيحة int's ، إذاً القائمة المرتبطة يمكن أن تكون معلنة كالتالي:

```
typedef struct nodetype
{
 int info;
 struct nodetype *next;
} node;
node *head;
```

تكوين قائمة فاضية Creating an empty list:

- في الإعلان السابق، المتغير الرأس head معلن كمؤشر للعقدة node.
- المتغير الرأس head حتى الآن لم يتم تهيئته.
- المتغير الرأس head يستخدم ليؤشر على العنصر الأول في القائمة list .
- لكون القائمة ستكون فاضيه empty في البداية، المتغير head يؤشر لقائمة فاضيه.

```
void createemptylist(node *head)
{
  head=NULL;
}
```

الإدخال في القائمة المرتبطة الأحادية Insertion in a Single Linked List :

لإدخال عنصر item في القائمة:

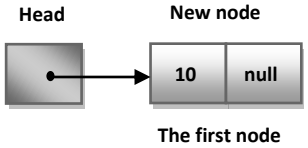
- يتم أولاً تكوين عقدة جديدة new node ،
- يحدد العنصر ليتم إدخاله في حقل المعلومات info field للعقدة الجديدة new node ،
- بعد ذلك يضع العقدة الجديدة new node في الموقع الملائم position بتعديل المؤشرات pointers .

الإدخال في القائمة يمكن أن يحدث في المواقع التالية:

1. الإدخال في بداية القائمة Insertion at the Beginning of the List :

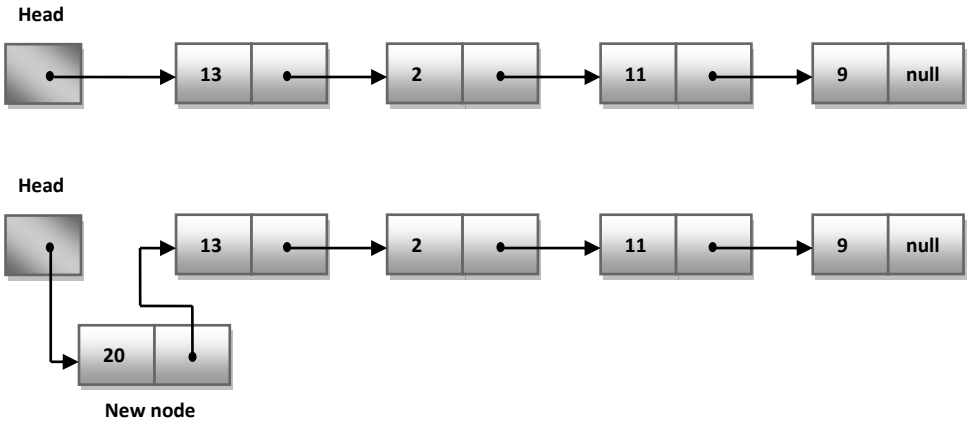
أولاً، نختبر إذا كانت القائمة المرتبطة linked list فاضية empty أم لا . إذا كانت فاضية empty، يتم إدخال العنصر item وهو بذلك يعتبر أول عنصر تم إدخاله في القائمة، وبذلك يكون هناك عنصر واحد فقط داخل القائمة. ويتم ذلك بإجراء الخطوات التالية:

- يتم تخصيص الـ null لحقل المؤشر next للعقدة الجديدة new node.
- يتم تخصيص العنوان address للعقدة الجديدة new node للرأس head.



إذا لم تكن القائمة فاضية empty، يتم إدخال العنصر item قبل أول عنصر في القائمة. ويتم إنجاز ذلك عن طريق الخطوات التالية:

- يتم تخصيص قيمة الرأس head لحقل المؤشر next للعقدة الجديدة new node.
- يتم تخصيص العنوان address للعقدة الجديدة new node للرأس head.



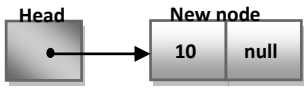
برمجياً، كلا الحالتين متكافئتين . وذلك لأن في الحالتين الخطوة الأول يتم فيها تخصيص قيمة الرأس head (null أو غير ذلك) لحقل المؤشر next للعقدة الجديدة new node.

```
void insertatbeginning(node *head, int item)
{
 node *newNode;

/* allocate memory for the new node and initialize the data in it*/
 newNode= malloc(sizeof(node));
 newNode->info=item;

 /* assign the value of head to the "next" of newNode*/
 newNode->next=head;

 /* assign the address of newNode to head */
 head=newNode;
}
```

2. الإدخال في نهاية القائمة Inserting at the End of the List :

أولاً، نختبر إذا كانت القائمة المرتبطة linked list فاضية empty أم لا . إذا كانت فاضية empty، يتم إدخال العنصر item وهو بذلك يعتبر أول عنصر تم إدخاله إلى القائمة، وبذلك يكون هناك عنصر واحد فقط داخل القائمة . ويتم إنجاز ذلك عن طريق الخطوات التالية:

- يتم تخصيص الـ null لحقل المؤشر next للعقدة الجديدة new node.
- يتم تخصيص العنوان address للعقدة الجديدة new node للرأس head.



The first node

إذا كانت القائمة ليست فاضية not empty ، يتم الوصول إلى العنصر الأخير في القائمة last item ، ثم يتم إدخال العقدة الجديدة new node كأخر عنصر last item في القائمة، ويتم ذلك بإجراء الخطوات التالية:

- يتم تخصيص الـ null لحقل المؤشر next للعقدة الجديدة new node.
- يتم تخصيص العنوان address للعقدة الجديدة new node لحقل المؤشر next للعقدة الأخيرة last node.

```
void insertatend(node *head, int item)
{
 node *newNode;
 newNode=malloc(sizeof(node));
 newNode->info=item;
 newNode->next=NULL;
 if(head==NULL)
   head=newNode;
 else
  {
    node *prev=head;
    while(prev->next!=NULL)
     prev=prev->next;
    prev->next=newNode;
  }
}
```

3. الإدخال بعد عنصر معطى Inserting after Given Element :

لإدخال عنصر جديد بعد عنصر معطى given element. أولاً، يتم تحديد الموقع للعنصر المعطى في القائمة وليكن loc ، يتم إدخال العنصر في القائمة بإجراء الخطوات التالية :

- يتم تخصيص حقل المؤشر next للعقدة المشار لها بواسطة loc لحقل المؤشر next للعقدة الجديدة new node.
- يتم تخصيص العنوان address للعقدة الجديدة new node لحقل المؤشر next للعقدة المشار لها بواسطة loc.

```
void insertafterelement(node *head, int item, node *loc)
{
 node *newNode;
 newNode=malloc(sizeof(node));
 newNode->info=item;
 newNode->next=loc->next;
 loc->next=newNode;
}
```

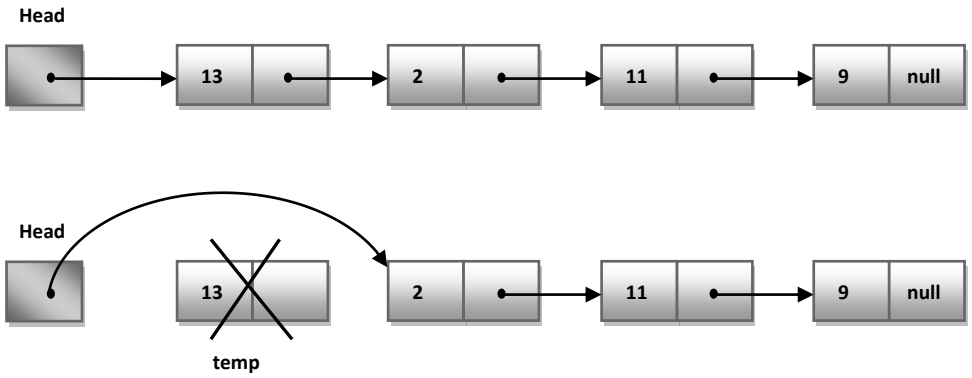
لإلغاء عنصر من القائمة المرتبطة الأحادية Deleting an Element from a Single Linked List :

لإلغاء عنصر من القائمة، أولاً يتم وضع المؤشرات pointers بشكل صحيح بحيث يتم إلغاء العقدة node من القائمة، العقدة الملغاة لا تزال تشغل موقع في الذاكرة memory ولتحرير هذا الموقع نستخدم دالة (free) . ويتم حذف العقدة node من القائمة من المواقع التالية:

1. الإلغاء من بداية القائمة Deleting from the Beginning of the List :

لإلغاء العنصر من بداية القائمة يتم ذلك بإجراء الخطوات التالية:

- يتم تخصيص قيمة الرأس head (العنوان address للعنصر الأول في القائمة) للمؤشر temp.
- يتم تخصيص قيمة حقل المؤشر next للعقدة الأولى للرأس head.
- يتم إلغاء تخصيص الذاكرة deallocate للعقدة المشار لها بواسطة temp.

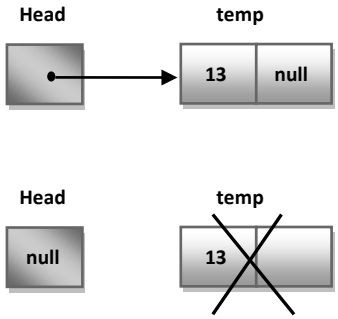



```
void deletefrombeginning( node *head)
{
    node *temp;
    if (head == NULL)
        return;
    else
    {
        temp = head;
        head = head ->next;
        free(temp);
    }
}
```

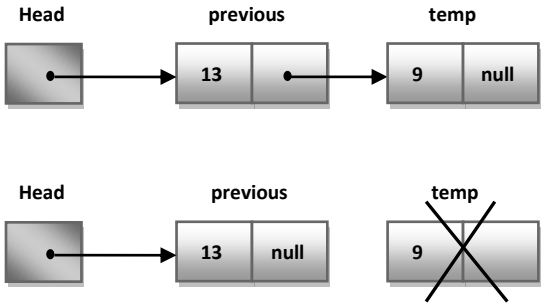
2. الإلغاء من نهاية القائمة Deleting from the End of the List :

للإلغاء من نهاية القائمة، لابد من الوصول إلى العنصر ما قبل الأخير في القائمة . بعد ذلك يمكن إلغاء العنصر الأخير last item بإجراء الخطوات التالية:

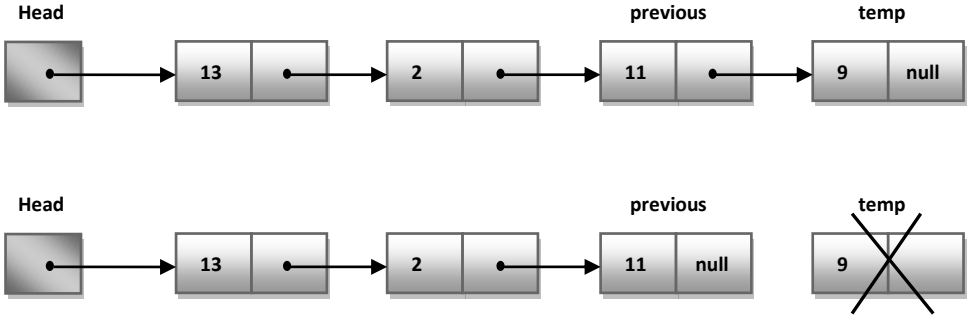
- يتم تخصيص حقل المؤشر next للعقدة ما قبل الأخيرة للمؤشر temp ,
- يتم تخصيص null لحقل المؤشر next للعقدة ما قبل الأخيرة في القائمة.
- يتم إلغاء تخصيص الذاكرة deallocate باستخدام دالة free للعقدة المشار لها بواسطة المؤشر temp .



(a)



(b)



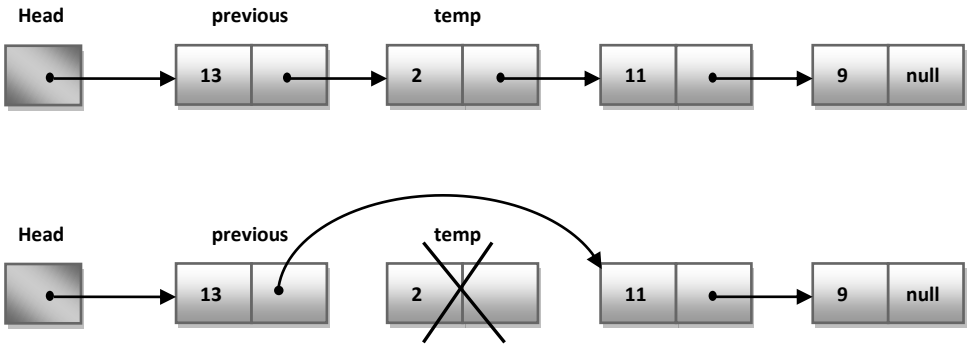
(c)

```
void deletefromend( node *head)
{
    node *temp, *prev;
    if (head == NULL)
        return;
    else if (head->next == NULL)
    {
        temp = *head;
        head = NULL;
        free(temp);
    }
    else
    {
        prev = head;
        temp = head->next;
        while (temp->next != NULL)
        {
            prev = temp;
            temp = temp->next;
        }
        prev->next = NULL;
        free(temp);
    }
}
```

3. الإلغاء بعد عنصر معطى Deleting after a Given Element :

لإلغاء العنصر بعد عنصر معطى prev ، يتم إجراء الخطوات التالية:

- يتم تخصيص حقل المؤشر next للعقدة المشار لها بواسطة prev للمؤشر temp .
- يتم تخصيص حقل المؤشر next للعقدة الملغاة لحقل المؤشر next للعقدة المشار لها بواسطة prev .
- يتم إلغاء تخصيص الذاكرة deallocate باستخدام دالة free للعقدة المشار لها بواسطة المؤشر temp .



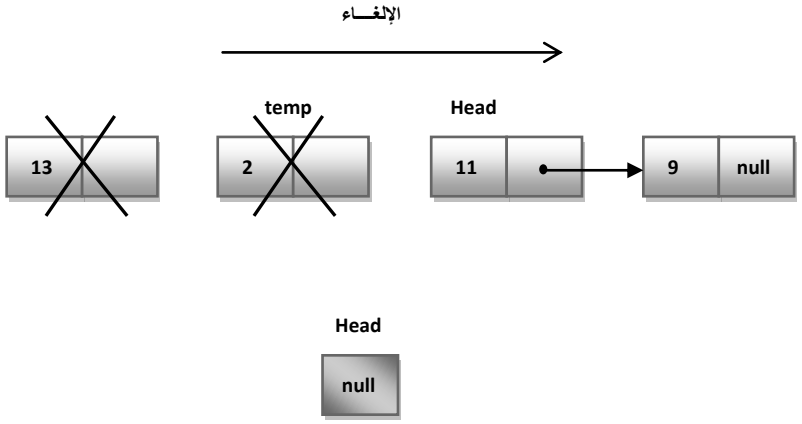

```
void deleteafterelement(node *head, node * prev)
{
    node *temp;
    if (prev->next==NULL)    /*element 'after' not found*/
        return;
    temp= prev->next;
    prev->next=temp->next;
    free(temp);
}
```

4. إلغاء القائمة بالكامل Deleting the Entire List :

قبل إنهاء البرنامج، لابد من إلغاء القائمة بالكامل وذلك لأن الذاكرة محجوزة من قبل القائمة list ولابد من إعادة الذاكرة المحجوزة لنظام التشغيل os . يمكن إنجاز هذه المهمة بإجراء الخطوات التالية:

- يتم تخصيص مؤشر الرأس head للمؤشر temp .
- يتقدم مؤشر الرأس head للعقدة التالية.
- يتم إلغاء تخصيص الذاكرة deallocate باستخدام دالة free للعقدة المشار لها بواسطة المؤشر temp .

يتم تكرار الخطوات السابقة حتى يتم إلغاء القائمة بالكامل.

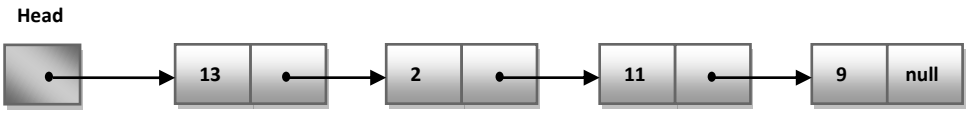



```
void deletelist(node *head)
{
    node *temp;
    while(head!=NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```

البحث في قائمة مرتبطة أحادية Searching a single Linked List :

نعبر القائمة من البداية، ونقارن كل عنصر في القائمة بالعنصر المعطى بواسطة المتغير temp والمطلوب البحث عنه.

item=11




```
node *searchunsortedlist(node *head, int item)
{
 while((head!=NULL) &&(head->info!=item))
 head=head->next;
 return head;
}
```